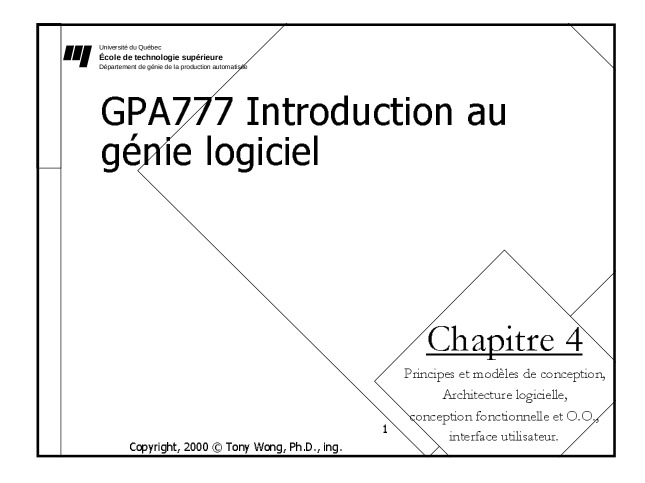


La conception (1)

- Est-ce que la conception ?
  - Un ensemble d'activités concentrées sur quatre aspects du développement:
    - Données
    - Architecture
    - Interfaces
    - composants
- Pourquoi est-elle si importante ?
  - Peut-on construire un édifice sans un plan de construction ?

2

La conception (2)

- Qui s'occupe de la conception ?
  - Au niveau de la conception des données et l'architecture:
    - Concepteurs spécialisés dans l'application des patrons de conception.
  - Au niveau des interfaces:
    - Concepteurs spécialisés dans les normes et standards et dans l'ergonomie du travail.
  - Au niveau des composants:
    - Concepteurs spécialisés dans la programmation informatique.

3

La conception (3)

- Quelles sont les étapes de conception ?
  - La conception débute par le modèle de travail dégagé lors de l 'analyse des exigences.
  - La conception consiste à transformer le résultat de cette analyse en 4 niveaux:
    - Structure de données
    - Architecture du système logiciel
    - Représentation des interfaces
    - Détail des composants
  - À chacun des niveaux, la conception doit tenir compte des principes et concepts appropriés.

La conception (4)

- Quel est le(s) livrable(s) de la conception ?
  - Il s'agit de la **spécification de la conception**.
  - Cette spécification comprend:
    - Le(s) modèle(s) des données
    - Le(s) modèle(s) de l'architecture
    - Le(s) modèle(s) des interfaces
    - Le(s) modèle(s) des composants
  - Ces modèles sont des livrables de la conception.

La conception (5)

- Comment déterminer l'exactitude de la conception ?
  - Valider chacun des livrables selon les points suivants:
    - La clarté
    - La correspondance
    - La complétude
    - La consistance
  - Ces points sont comparés avec les résultats de l'analyse des exigences et vice versa.

Principes de conception (1)

- La conception est à la fois un processus et un modèle.
- Le processus de conception est une séquence d'activités qui permet aux concepteurs de décrire tous les aspects du logiciel à construire.
- Le modèle de conception est un plan qui donne un nombre de vues différentes du logiciel à construire.

7

Principes de conception (2)

- Il existe un ensemble de principes de base qui aident les concepteurs dans le processus de conception:
  - Les concepteurs doivent considérer les approches alternatives avant de prendre une décision finale.
  - La conception doit être basée sur les résultats de l'analyse des exigences. On doit pouvoir expliquer les détails de la conception en remontant à l'analyse des exigences.

8

Principes de conception (3)

- Dans la mesure du possible, la conception doit reposer sur des patrons déjà existants. Ne pas réinventer la roue !
- La structure de la conception doit calquer, dans la mesure du possible, sur la structure du problème à résoudre.
- La conception doit montre l'uniformité et l'intégrité.
  - Le style et le format de travail doivent être définis. (Uniformité)
  - Les interfaces entre les composants de conception sont bien définies. (Intégrité)

9

Principes de conception (4)

- La conception doit être structurée en fonction de la possibilité de changement.
- La conception doit être révisée pour minimiser les erreurs.

Enfin,

- **La conception n'est pas le codage, le codage n'est pas la conception !**

- Il faut suivre ces quelques principes de conception dans vos projets de développement.

10

Concepts pratiques (1)

- Il existe un ensemble de concepts développés qui facilitent le travail de la conception.
- Ces concepts donnent des réponses possibles à des questions importantes de la conception:
  - Quels sont les critères utilisés dans la séparation du logiciel en composants individuels ?
  - Comment dégager le détail des fonctions et des données à partir de la représentation du logiciel ?
  - Quels sont les critères qui définissent la qualité technique d'une conception logicielle ?

11

Concepts pratiques (2)

- Les concepts présentés ci-dessous, lorsque appliqués correctement, mèneront vers une conception robuste et efficace:
  - Abstraction
    - Ce concept est utilisé pour donner une solution modulaire à des problèmes.
    - Il s 'agit d'un concept couramment utilisé dans le domaine du génie.
    - La solution est présentée en niveaux. Le plus haut niveau est l'abstraction générale, les niveaux subséquents présentent de plus en plus de détails.

12

Concepts pratiques (3)

- Abstraction (suite)
  - Nous pouvons appliquer l'abstraction à des procédures, à des fonctions, à des données et à des mécanismes de contrôle.
- Raffinement
  - Il s'agit d'une stratégie de conception haut-vers-bas (*top-down*) qui consiste à raffiner successivement la définition d'une procédure, fonction, des données et des mécanismes de contrôle.
  - La différence entre l'abstraction et le raffinement est que ce dernier doit aboutir à des énoncées de langage de programmation.

13

Concepts pratiques (4)

- Modularité
  - Le logiciel est divisé en composants distincts appelés **modules**.
  - L'intégration des modules doit satisfaire les exigences du logiciel.
  - Il s'agit d'une conséquence de la stratégie « diviser pour régner » qui est appliquée partout dans le domaine de l'ingénierie.
  - Il est aussi le concept central de la conception moderne !

14

Concepts pratiques (5)

- Modularité (suite)
  - Comment évaluer la modularité de notre conception ?
    - Vérifier la <u>décomposabilité modulaire</u>. Il doit exister une manière systématique de décomposer le problème à résoudre en sous-problèmes.
    - Vérifier la <u>composabilité modulaire</u>. Il doit pouvoir intégrer, dans notre système logiciel, des composants réutilisables.
    - Vérifier la <u>compréhension modulaire</u>. Un module doit être auto-suffisant (*standalone*).
    - Vérifier la <u>continuité modulaire</u>. Les changements d'exigences doivent entraîner des changements dans un petit nombre de modules.
    - Vérifier la <u>protection modulaire</u>. Les conditions d'erreurs à l'intérieur d'un module ne doivent pas propager dans l'ensemble des modules.

15

Concepts pratiques (6)

- Architecture logicielle
  - Elle donne la structure générale du logiciel.
  - Elle montre les relations hiérarchiques des modules (composants) du logiciel.
  - Elle sert de cadre de travail (*framework*) pour la conception détaillée des modules (composants).
  - 5 types de base sont couramment utilisés:

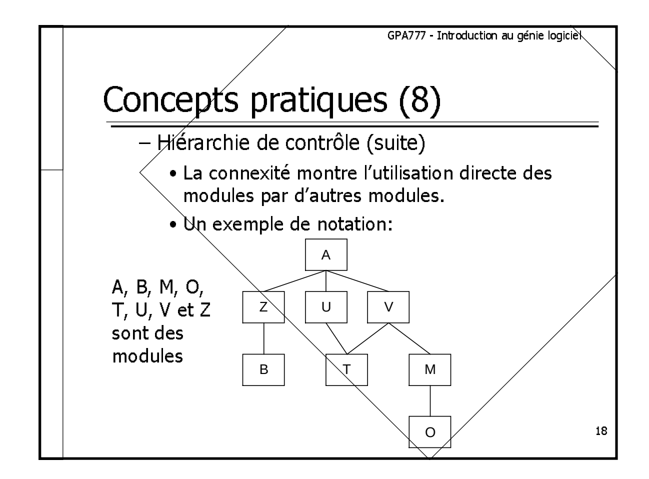
À titre d'exemple
    - Modèle structurel
    - Modèle cadre de travail
    - Modèle dynamique
    - Modèle processus
    - Modèle fonctionnel

Concepts pratiques (7)

- Hiérarchie de contrôle
  - Ne s 'applique à des architectures qui possèdent une hiérarchie.
  - Aussi connue sous le nom de « structure du programme » et elle décrit la relation de contrôle qui existe entre les modules du système logiciel.
  - Elle présente la **visibilité** et la **connexité** d'une architecture logicielle.
  - La visibilité montre la possibilité d'utilisation des modules par d'autres modules.
    - Ex: Dans l'approche orientée objet, un objet peut hériter un nombre de données mais n'utilise qu'un sous ensemble.

Concepts pratiques (8)

- Hiérarchie de contrôle (suite)
  - La connexité montre l'utilisation directe des modules par d'autres modules.
  - Un exemple de notation:



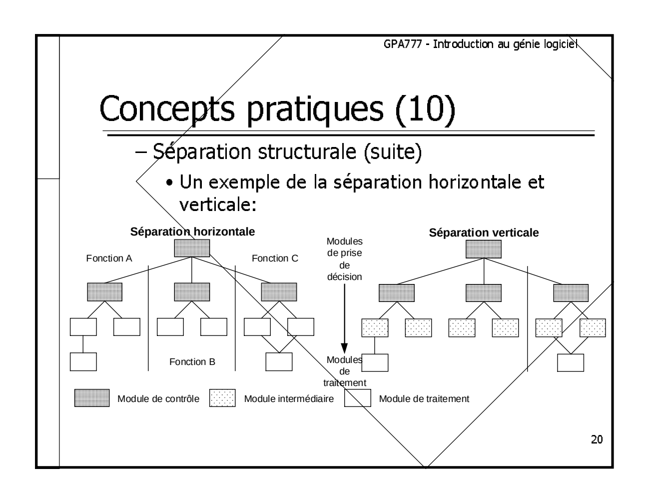
Concepts pratiques (9)

– Séparation structurale
- Ne s'applique qu'à des architectures qui possèdent une hiérarchie.
- La hiérarchie de contrôle peut être séparée horizontalement ou verticalement.
- Séparation horizontale:
  – Regrouper dans une branche distincte chaque fonction majeure du programme.
  – Le contrôle est distribué entre les branches horizontalement.
- Séparation verticale:
  – Le contrôle est distribué de haut vers le bas. La prise de décision est effectuée par les modules de niveaux supérieurs. Le traitement est réalisé par des modules de niveaux inférieurs.

19

Concepts pratiques (10)

– Séparation structurale (suite)
- Un exemple de la séparation horizontale et verticale:



20

Concepts pratiques (11)

– Structure de données
- Il s'agit d'une représentation logique des relations entre les éléments d'information utiles à un logiciel.
- Il faut toujours déterminer la représentation la plus efficace que possible puisque cette représentation aura des répercussion non négligeable sur les algorithmes et l'efficacité des traitements.

– Procédure du logiciel
- Elle montre la séquence des traitements, les points de décision, les opérations répétitives, etc.

Les ordinogrammes servent à exprimer la procédure du logiciel

21

Concepts pratiques (12)

- Masquage de l'information
  - Ce principe consiste à établir des modules dont les informations contenues ne sont accessibles que par des modules qui en ont besoin.
  - Le but est de favoriser la conception de modules indépendants collaborant par des échanges d'information.
  - L'approche orientée objet est la réalisation de ce principe de conception.

22

Heuristiques de conception (1)

- Il existe un ensemble d'heuristiques qui facilitent la modularité:
  - Créer des versions améliorées de la hiérarchie de contrôle (structure du programme).
  - Unir ou séparer les modules afin de réduire le couplage qui existe entre les modules.
  - Réduire le nombre de niveaux dans la hiérarchie de contrôle.
  - Éviter l'étalement horizontal des modules.
    - ~~Les modules sont tous au même niveau.~~

23

Heuristiques de conception (2)

- Encadrer la responsabilité des modules.
  - Un module est responsable de son domaine et pas plus.
- Réduire la complexité des interfaces entre modules.
- Les modules ne doivent pas contenir des informations redondantes ou dupliquées.

24

Modèle de conception (1)

- Les principes et concepts présentés sont nécessaires pour la création d'un modèle de conception.
- Le modèle de conception comprend la représentation de données, de l'architecture, des interfaces et des composants.
- C'est à partir de ce modèle que le codage est réalisé.

25

Modèle de conception (2)

- Le modèle de conception est pyramidal:

Conception des composants
Conception des interfaces
Conception de l'architecture
Conception des données

26

Documentation (1)

- La spécification de conception comprend:
  - Spécification des données
    - Spécification de la structure des bases de données.
    - Spécification des formats de fichiers.
    - Spécification des structures de données du programme.
  - Spécification de l'architecture
    - Montrer comment l'architecture du logicielle a été dérivée à partir des exigences.
    - Présenter la hiérarchie des modules.

27

Documentation (2)

- Spécification des interfaces
  - Présenter les interfaces entre les modules du logiciel.
  - Montrer les interfaces utilisateurs (GUI).
- Spécification des composants
  - Présenter le rôle de chacun des modules.
  - Expliquer les relations logiques entre ces modules.
- Référence croisée entre les spécifications de conception et les exigences du logiciel.
  - Présenter sous forme d'une matrice de correspondance.

Documentation (3)

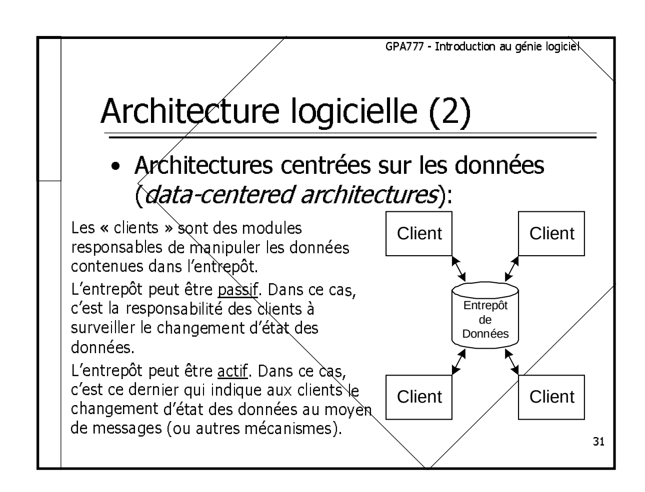
- Référence croisée entre les spécifications de conception et les exigences du logiciel.

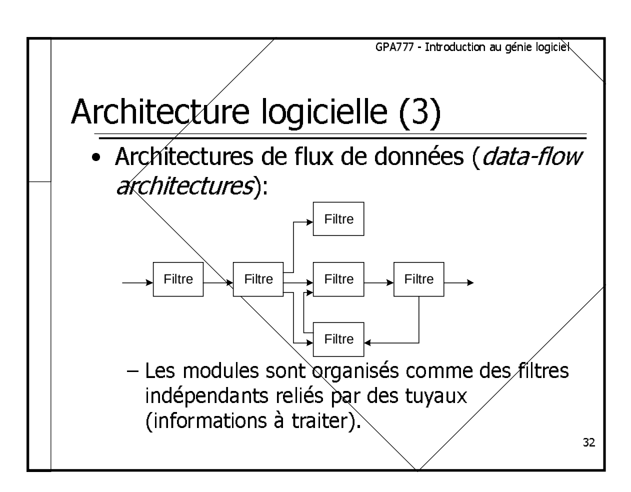
	Composant 1	Composant 2	Composant 3	Composant 4	Composant 5
Exigence 1	X		X		
Exigence 2		X	X		
Exigence 3				X	
Exigence 4	X				
Exigence 5					X
Exigence 6				X	

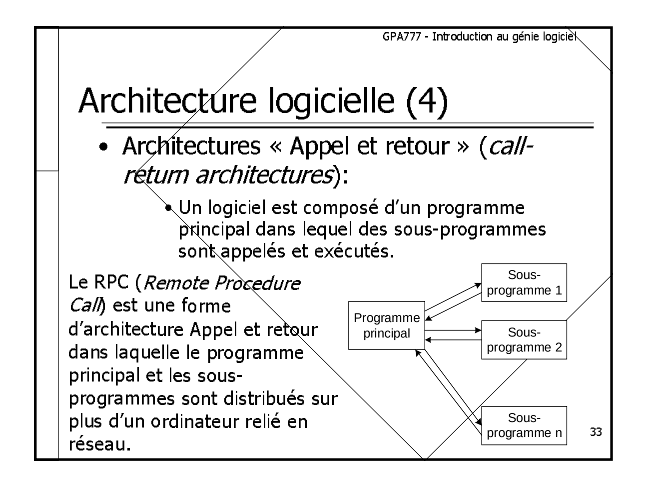
- Cette matrice donne un résumé de l 'importance des modules pour la satisfaction des exigences du logiciel.
- Vous devez indiquer clairement le nom des modules et le type des exigences.

Architecture logicielle (1)

- Il existe une panoplie d'architectures qui ont déjà été établies.
- Donc, au lieu d'inventer une toute nouvelle architecture, il est plus économique d'en adopter une ou d'en modifier une déjà existante.
- Voici une brève présentation de quelques unes les plus utilisées.


GPA777 - Introduction au génie logiciel
Architecture logicielle (2)
• Architectures centrées sur les données (*data-centered architectures*):
Les « clients » sont des modules responsables de manipuler les données contenues dans l'entrepôt.
L'entrepôt peut être passif. Dans ce cas, c'est la responsabilité des clients à surveiller le changement d'état des données.
L'entrepôt peut être actif. Dans ce cas, c'est ce dernier qui indique aux clients le changement d'état des données au moyen de messages (ou autres mécanismes).
Client
Client
Entrepôt de Données
Client
Client
31


GPA777 - Introduction au génie logiciel
Architecture logicielle (3)
• Architectures de flux de données (*data-flow architectures*):
Filtre
Filtre
Filtre
Filtre
Filtre
Filtre
– Les modules sont organisés comme des filtres indépendants reliés par des tuyaux (informations à traiter).
32


GPA777 - Introduction au génie logiciel
Architecture logicielle (4)
• Architectures « Appel et retour » (*call-return architectures*):
• Un logiciel est composé d'un programme principal dans lequel des sous-programmes sont appelés et exécutés.
Le RPC (*Remote Procedure Call*) est une forme d'architecture Appel et retour dans laquelle le programme principal et les sous-programmes sont distribués sur plus d'un ordinateur relié en réseau.
Programme principal
Sous-programme 1
Sous-programme 2
Sous-programme n
33

Architecture logicielle (5)

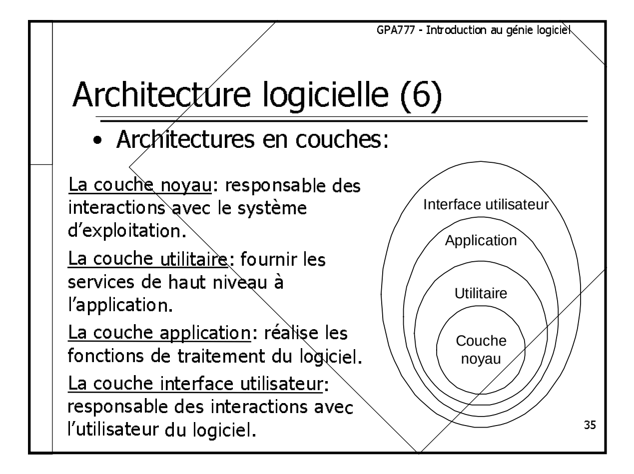
- Architectures orientées objet:
  - Ces architectures encapsulent les données et les opérations dans des composants (objets).
  - La communication et la coordination sont réalisées à l'aide de messages.

Nous pouvons aussi adapter les architectures présentées précédemment pour les systèmes orientés objet.

Architecture logicielle (6)

- Architectures en couches:

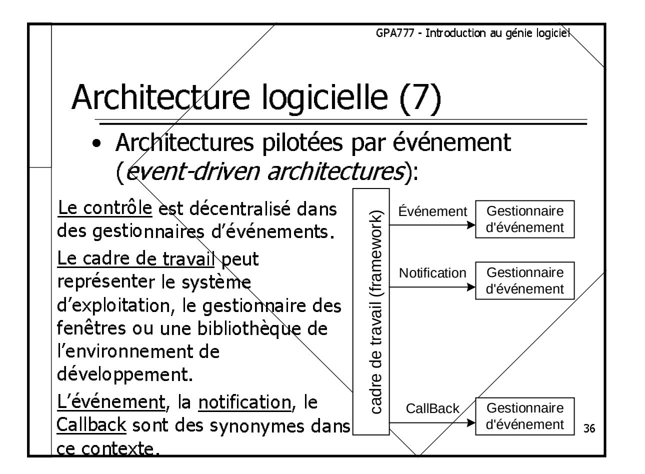
La couche noyau: responsable des interactions avec le système d'exploitation.
La couche utilitaire: fournir les services de haut niveau à l'application.
La couche application: réalise les fonctions de traitement du logiciel.
La couche interface utilisateur: responsable des interactions avec l'utilisateur du logiciel.



Architecture logicielle (7)

- Architectures pilotées par événement (*event-driven architectures*):

Le contrôle est décentralisé dans des gestionnaires d'événements.
Le cadre de travail peut représenter le système d'exploitation, le gestionnaire des fenêtres ou une bibliothèque de l'environnement de développement.
L'événement, la notification, le Callback sont des synonymes dans ce contexte.



Architecture logicielle (8)

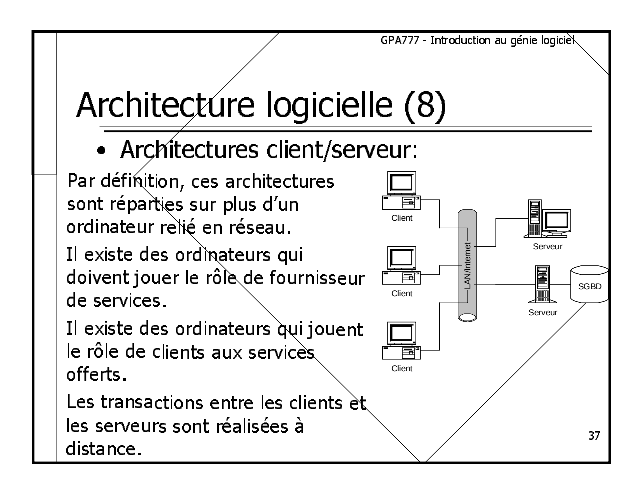
- Architectures client/serveur:

Par définition, ces architectures sont réparties sur plus d'un ordinateur relié en réseau.

Il existe des ordinateurs qui doivent jouer le rôle de fournisseur de services.

Il existe des ordinateurs qui jouent le rôle de clients aux services offerts.

Les transactions entre les clients et les serveurs sont réalisées à distance.



Architecture logicielle (9)

- Comment déterminer l'architecture qui convient à notre application ?
- Utiliser la méthode ATAM (*Architecture trade-off Analysis Method*) [KAZM98].
- Dans cette méthode effectue une évaluation itérative de la qualité de l'architecture en fonction des exigences du logiciel.
- ATAM est une méthode qualitative et non quantitative.

Architecture logicielle (10)

- Les activités associées à la méthode ATAM:
  - Rassembler les cas d'utilisation (voir analyse orientée objet).
  - Décrire le style d'architecture préconisé pour l'application selon les points de vue:
    - Point de vue module: Est-ce que l'architecture préconisée permet la création de modules convenables ? L'architecture préconisée permet-elle le masquage d'information ?
    - Point de vue processus: Quelle est la performance du système si l'on adopte l'architecture préconisée ?
    - Point de vue du flux de données: Est-ce que cette architecture convient aux exigences fonctionnelles du logiciel ?

Architecture logicielle (11)

- Évaluer les attributs de qualité isolément.
  - Ces attributs sont:
    - Fiabilité
    - performance
    - sécurité
    - flexibilité
    - entretien facile
    - vérifiabilité
    - portabilité
    - réutilisation
    - interopérabilité
  - Accorder une importance relative à ces attributs selon les besoins et exigences du logiciel.

Architecture logicielle (12)

- Identifier les « points sensibles » de l'architecture préconisée.
  - On peut découvrir les points sensibles d'une architecture en effectuant des changements mineurs et en vérifiant leur impact sur les attributs de qualité.
  - Les points sensibles d'une architecture sont les changements qui provoquent une variation importante sur les attributs de qualité.
- Critiquer l'architecture préconisée en se basant sur les points sensibles relevés.

Architecture logicielle (13)

- Cette critique s'appuiera sur l'effet des points sensibles d'une architecture sur les attributs de qualité désirés.
  - Par exemple, la performance de l'architecture client/serveur est hautement sensible au nombre de serveurs disponibles. Par contre, la sécurité d'une telle architecture varie inversement proportionnel au nombre de serveurs.
- Éliminer les architectures envisagées en fonction de la critique. Retenir celle qui présente moins d'effet négatif sur les attributs de qualité désirés.

Conception structurée (1)

- L'analyse structurée produit un SRS qui détaille les exigences du logiciel.
- Les diagrammes DFD, CFD issus de l'analyse structurée peuvent être associés à une architecture via une procédure de transformation.
- Les pages qui suivent présentent la méthode d'association par transformation (*Transform mapping*) pour la conception logicielle.

43

Conception structurée (2)

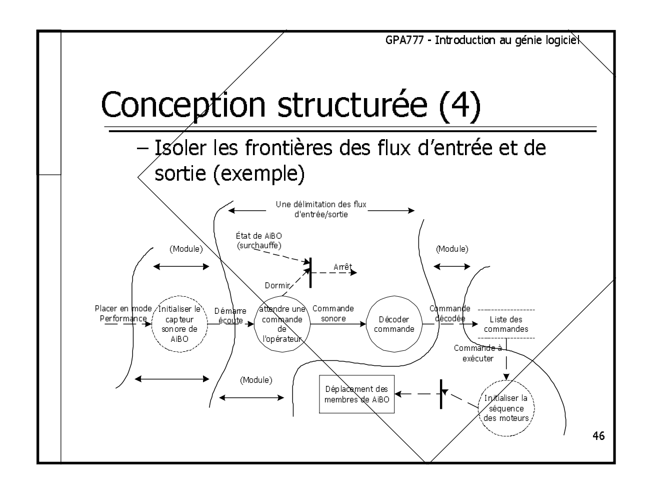
- Voici les étapes de la méthode d'association par transformation:
  - Révision du modèle fondamental du système logiciel.
    - Lecture et réévaluation du document SRS et surtout les diagrammes DFD.
  - Révision et raffinement des diagrammes DFD.
    - Vérifier que ces diagrammes contiennent suffisamment d'information pour la conception en tenant compte de la modularité éventuelle.
    - Apporter des ajouts ou modifications nécessaires.

44

Conception structurée (3)

- Isoler les frontières des flux d'entrée et de sortie.
  - C'est-à-dire, délimiter le flux des données d'entrée/sortie autour d'un ou plusieurs processus de traitement (les cercles dans un DFD).
  - L 'objectif est de dégager les modules appropriés à partir des DFD.
  - Délimiter le flux de la même façon pour tous les diagrammes DFD.
  - Attention. Il peut exister plus d'un module par diagramme DFD et un DFD peut ne pas contenir de module.

45



GPA777 - Introduction au génie logiciel

Conception structurée (5)

- Appliquer la séparation verticale.
  - Assigner la responsabilité de décision à des modules contrôleurs selon le schème de la séparation verticale (page 19-20).
  - Associer la responsabilité de traitement à des modules « travailleurs ».
- Associer les processus de traitement aux modules.
  - Chaque processus (bulle des DFD) est associé à un module du logiciel.
  - Appliquer le concept de modularité et de masquage d'information pour réaliser cette association.

47

GPA777 - Introduction au génie logiciel

Conception structurée (6)

- Raffiner l'association par transformation par application des heuristiques de conception et en tenant compte de l'architecture logicielle choisie.
- Continuer le processus avec les CFD.
- Continuer le raffinement des étapes précédentes.

48

Conception structurée (7)

- Pour la spécification des données, il suffit d'utiliser les données dégagées de l'analyse structurée (analyse des données).
  - Cependant, il est nécessaire de traduire les données de l'analyse en terme d'éléments de base de données, de structure de données, etc.
  - Par exemple: « Liste de nom » est traduit en tableau de chaînes de caractères MBCS (Multi-Byte Caracter Set) de 1024 caractères.

Conception structurée (8)

- Pour la spécification des interfaces, utiliser les frontières des flux d'entrée/sortie dégagées précédemment.
  - Étudier les frontières et noter les données associées aux flux d'entrée/sortie.
  - Établir l'interface des modules en fonction de ces données communiquées.
  - L'interface des modules est présentée sous forme de pseudo-code ou ordinogrammes.

Conception structurée (9)

- Pour la spécification des composants, elle consiste à définir:
  - Le rôle et la responsabilité des modules.
  - Le principe de fonctionnement des modules
  - Le comportement dynamique des modules.
  - Les algorithmes utilisés.
  - Cette spécification est présentée sous forme de pseudo-code ou ordinogrammes.

Conception structurée (10)

- Enfin, les diagrammes STD (diagrammes d'états) de l'analyse structurée servent à réaliser la coordination de l'exécution des modules du logiciel.
  - On doit déduire à partir des STD la séquence des activités du logiciel menant à la solution du problème (ou à l'accomplissement d'un objectif).
  - Les états contenus dans les STD peuvent être explicites ou implicites.

Les états sont représentés par des variables spécifiques

Les états sont déduits indirectement à l'aide d'un ensemble de variables

52

Conception structurée (11)

- Voici un exemple où les états du logiciel sont apparents.

Quelle est la représentation de ces états ?

53

Conception orientée objet (1)

- La conception O.O. est divisée en deux grands processus:
  - Le processus de conception système.
  - Le processus de conception d'objet.
- La conception système comprend les activités suivantes:
  - Séparation du modèle d'analyse en sous-systèmes. (Contenu dans le SRS)
    - Cette séparation est implicite dans le modèle d'analyse.

54

Conception orientée objet (2)

- Séparation du modèle d'analyse en sous-systèmes (suite).
  - Un sous-système doit avoir une interface bien définie qui sert à communiquer avec les autres sous-systèmes du logiciel.
  - Un sous-système doit posséder un petit nombre de classes qui communiquent avec les autres sous-systèmes (interfaces). Le reste des classes du sous-système doivent communiquer entre elles.
  - Il faut conserver le nombre de sous-systèmes au minimum.

55

Conception orientée objet (3)

- Établissement des couches d'abstraction en terme de fonctionnalité (architecture).
  - Par exemple, une architecture à 4 couches:

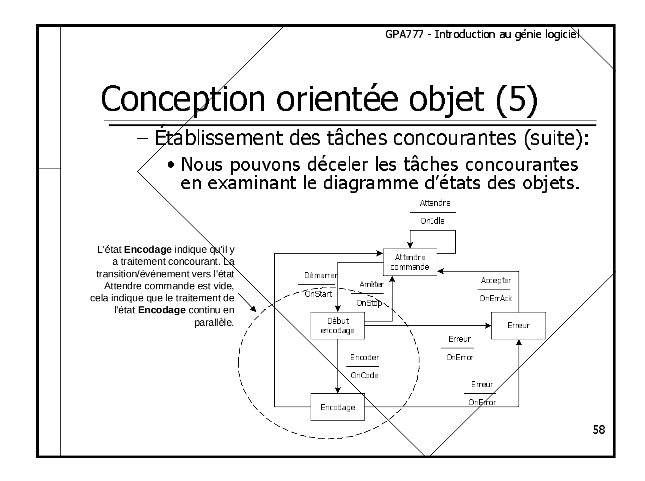
Couche de présentation	Sous-systèmes associés à l'interface utilisateur
Couche d'application	Sous-systèmes qui réalisent des traitements de données
Couche de formatage des données	Sous-systèmes qui préparent les données pour le traitement
Couche de base de données	Sous-systèmes associés à la gestion des données

56

Conception orientée objet (4)

- Établissement des tâches concourantes:
  - Cette activité consiste à identifier les sous-systèmes et les classes qui sont actifs en même temps.
  - Il y a aura traitements concourants s'il existe des sous-systèmes ou classes qui doivent répondre aux événements d'une manière asynchrone et au même moment.
  - Dans le cas de traitements concourants, nous pouvons:
    - Allouer les sous-systèmes ou classes dans des processus différents
    - Allouer les sous-systèmes ou classes dans le même processeur en utilisant la capacité concourante du système d 'exploitation.

57



GPA777 - Introduction au génie logiciel

Conception orientée objet (6)

– Établissement la gestion des tâches concourantes:
- Il s 'agit d'établir une liste contenant les informations suivantes:
  – Nom de la tâche
  – Description
  – Priorité de la tâche (ex: bas, médium, élevé)
  – Services offerts
  – Invoqué par (donner la façon dont cette tâche est initiée et exécutée)
  – Communication (les données reçues et envoyées par cette tâche)
- Cette liste facilitera la programmation qui sera réalisée subséquemment.

59

GPA777 - Introduction au génie logiciel

Conception orientée objet (7)

– Gestion des données:
- Il s'agit d'un travail à deux niveaux:
  – La gestion des données critiques au fonctionnement de l'application.
  – La création d'infrastructure de stockage pour les objets manipulés.
- Dans le premier cas, on utilisera les structures de données classiques et/ou avancées.
  – Elles sont normalement disponibles dans des bibliothèques standards du langage de programmation O.O.
- Dans le second cas, on doit recourir à la disposition de l'environnement de développement.
  – Par exemple, le « sérialisation » des objets de MFC et de Java.

60

Conception orientée objet (8)

- Communication inter-sous-systèmes:
  - On doit établir le schème de communication entre les sous-systèmes de l'application.
  - On considère une communication entre deux sous-systèmes comme un **contrat** de travail.
  - Comme tout contrat de travail, il comporte un nom et nous devons se mettre en accord pour la définition.
  - Une fois le contrat défini, il ne doit y avoir de changement sans le consentement des sous-systèmes impliqués.

Fin du processus de conception système

Conception orientée objet (9)

- La conception d'objet comprend les activités suivantes:
  - Description des objets par le protocole et par l'implantation.
    - La description protocolaire établie l'interface publique de l'objet sous forme de message et de commentaire:
      - message(encodeur)--> lire: retourne fichier.donnees
      - message(encodeur)--> fin: envoie FIN_CONVERSION
      - message(encodeur)--> debut: regle encodeur.dflag
      - etc.
    - La description protocolaire ne décrit pas la portion privée de l'objet.

Conception orientée objet (10)

- Description des objets par le protocole et par l'implantation (suite).
  - La description d'implantation décrit les éléments internes de l'objet.
  - Cette description comprend:
    - Le nom de l'objet et son appartenance (sa classe).
    - La spécification de ses structures de données privées.
    - La description de ses opérations en pseudo-code ou en ordinogrammes.
  - La description d'implantation doit aussi englober les services/messages présentés dans la description protocolaire.

Conception orientée objet (11)

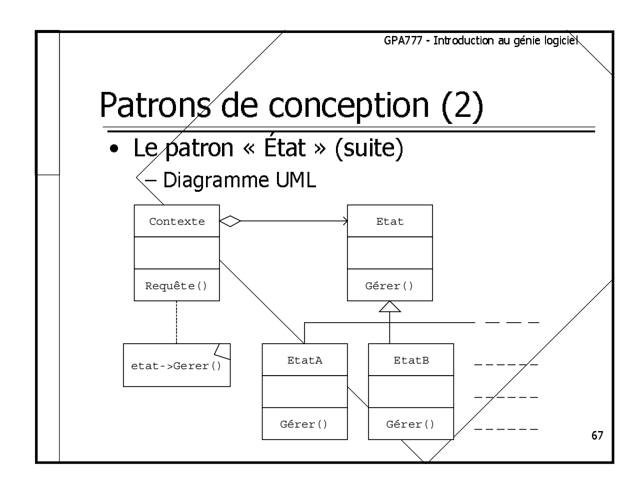
- Conception des algorithmes et structures de données.
  - Cette activité est semblable à celle de la conception structurée.
  - Cependant, il est possible qu'un algorithme de traitement soit réalisé par plusieurs objets en collaboration. Autrement dit, le contrôle de l'exécution de l'algorithme est répartie.
  - Attention, il faut toujours concevoir les structures de données en même temps que les algorithmes qui les utilisent.
  - Une mauvaise conception des structures de données peut entraîner une perte d'efficacité des algorithmes.

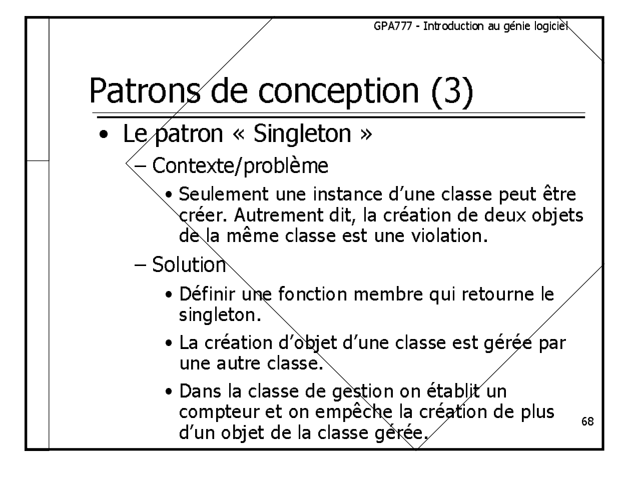
Conception orientée objet (12)

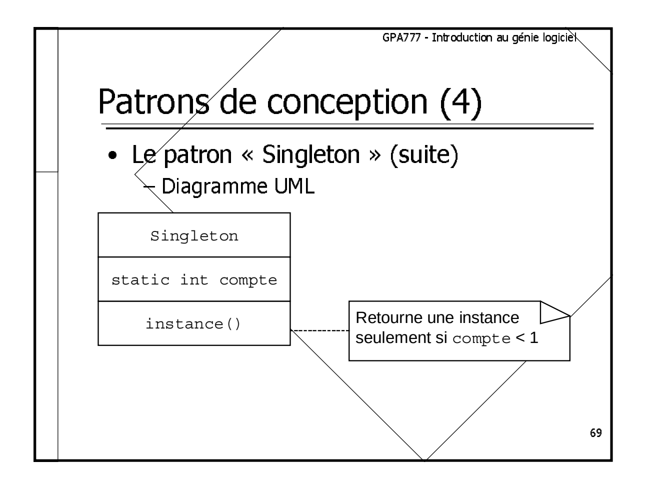
- Détermination des patrons de conception:
  - Au lieu de réinventer la roue, on peut utiliser les patrons de conception « préfabriqués » pour simplifier le travail.
  - Les patrons de conception sont des constructions d'objet qui répondent à des besoins spécifiques de la conception logicielle.
  - Ces patrons sont issus de l'expérience des concepteurs logiciels et ils sont réutilisables et applicables dans des contextes semblables.
  - Chaque patron de conception porte un nom propre et on peut les référer par leur nom.

Patrons de conception (1)

- Le patron « État »
  - Contexte/problème
    - Le comportement d'un objet est dépendant de son état. La logique conditionnelle n'est pas applicable à cause de sa complexité.
  - Solution
    - Créer une classe pour chaque état qui influence le comportement de l'objet.
    - En utilisant le polymorphisme, assigner les opérations à chaque classe-état pour gérer le comportement.
    - Lorsqu'un message est reçu par l'objet, il passe le message à l'un des objets-état.


GPA777 - Introduction au génie logiciel
Patrons de conception (2)
• Le patron « État » (suite)
– Diagramme UML
Contexte
Requête()
Etat
Gérer()
etat->Gerer()
EtatA
Gérer()
EtatB
Gérer()
67


GPA777 - Introduction au génie logiciel
Patrons de conception (3)
• Le patron « Singleton »
– Contexte/problème
• Seulement une instance d'une classe peut être créer. Autrement dit, la création de deux objets de la même classe est une violation.
– Solution
• Définir une fonction membre qui retourne le singleton.
• La création d'objet d'une classe est gérée par une autre classe.
• Dans la classe de gestion on établit un compteur et on empêche la création de plus d'un objet de la classe gérée.
68


GPA777 - Introduction au génie logiciel
Patrons de conception (4)
• Le patron « Singleton » (suite)
– Diagramme UML
Singleton
static int compte
instance()
Retourne une instance seulement si compte < 1
69

Patrons de conception (5)

- Le patron « Proxy à distance »
  - Contexte/problème
    - Le système doit communiquer avec un composant sur une autre machine. Qui doit être le responsable ?
  - Solution
    - Créer une classe locale qui représente le composant externe.
    - La doter de capacités de communication avec le composant réel à distance.
    - Le système local communique avec une instance de cette classe comme s'il communiquerait avec le composant réel.

Patrons de conception (6)

- Le patron « Proxy »
  - Contexte/problème
    - Accès direct à un composant n'est pas désirable. Que faire ?
  - Solution
    - Créer une classe qui représente le composant externe.
    - La doter de capacités de communication avec le composant réel.
    - Le système local communique avec une instance de cette classe comme s'il communiquerait avec le composant réel.

Patrons de conception (7)

- Le patron « Enveloppe »
  - Contexte/problème
    - Un module ou une bibliothèque n 'a pas été conçu selon l'approche orientée objet. Que faire ?
  - Solution
    - Créer une classe qui représente représente le module ou la bibliothèque.
    - La doter d'une interface publique qui offre les services du module ou bibliothèque.
    - La doter d'une implantation qui fait appel aux fonctions du module ou bibliothèque.
    - L'application doit utiliser cette classe.

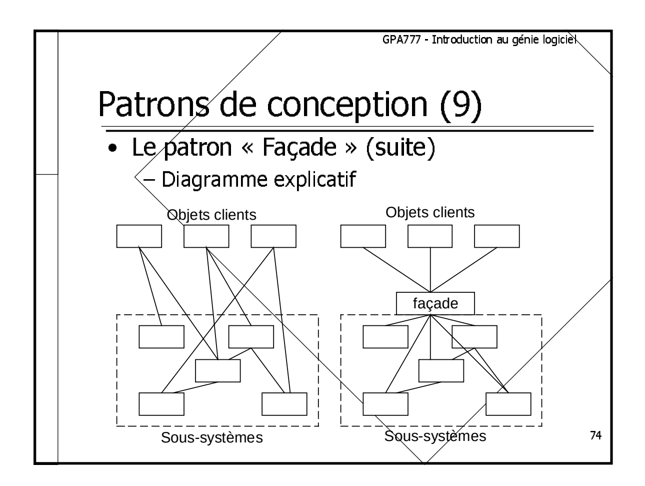
Patrons de conception (8)

- Le patron « Façade »
  - Contexte/problème
    - Une interface unifiée qui regroupe un ensemble d'interfaces de sous-systèmes.
  - Solution
    - Créer une seule classe qui présente dans son interface publique les services des sous-systèmes.
    - Un appel de ces services est redirigé vers le sous-système approprié par cette classe unificatrice.

73

Patrons de conception (9)

- Le patron « Façade » (suite)
  - Diagramme explicatif

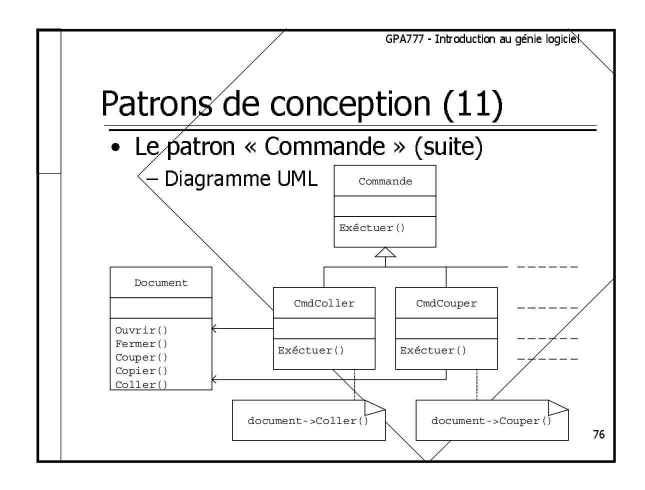


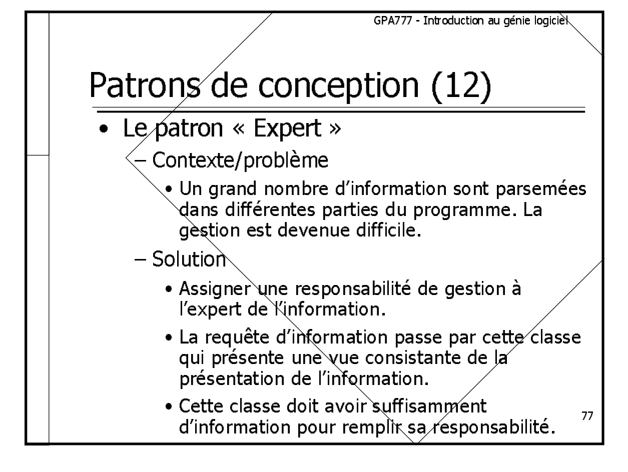
74

Patrons de conception (10)

- Le patron « Commande »
  - Contexte/problème
    - Un grand nombre de requêtes ou commandes peuvent être reçues par un objet. Nous devons réduire sa responsabilité et permet l 'ajout de nouvelles requêtes ou commandes.
  - Solution
    - Pour chaque commande, créer une classe responsable de son traitement.
    - Utiliser l'héritage pour lier les classes de traitement de commande à la classe de départ.

75





GPA777 - Introduction au génie logiciel

Patrons de conception (13)

- Le patron « Créateur »
  - Contexte/problème
    - Comment détermine-t-on la responsabilité de création des objets ?
  - Solution
    - Assigner la responsabilité de création d'objets de classe A par la classe B si l'une des énoncées suivantes est vraie:
      - B possède un relation d'agrégation avec A
      - B contient A
      - B enregistre les instances de A
      - B utilise intimement A
      - B possède les paramètres initiaux de A.

78

Patrons de conception (14)

- Le patron « Fabrication pure »
  - Contexte/problème
    - Il est nécessaire d'obtenir un couplage faible et une grande cohésion entre les classes. Mais ce n'est pas possible avec les résultats d 'analyse obtenus.
  - Solution
    - Créer une ou des classes purement imaginaires qui ne sont pas dans le domaine du problème.
    - Leur assigner des responsabilités qui peuvent réduire le couplage et augmenter la cohésion.
    - Ces classes ne modélisent aucun élément du problème et sont artificielles.

79

Patrons de conception (15)

- Le patron « Indirection »
  - Contexte/problème
    - Il existe un grand couplage entre deux classes (et plus). Que faire ?
  - Solution
    - Créer une classe intermédiaire qui agit comme agent entre les classes dont le couplage est fort.
    - Les transactions entre ces classes doivent passer par l'agent intermédiaire.

80

Patrons de conception (16)

- Le patron « Publication-souscription »
  - Contexte/problème
    - Il existe un grand couplage entre plusieurs classes. Que faire ?
  - Solution
    - Créer une classe de gestion d'événements (un *eventManager* par exemple).
    - Les objets publient leur message à la classe de gestion.
    - Cette dernière annonce les messages aux classes souscrits.
    - Il existe alors un découplage entre les objets qui publient et ceux qui sont souscrits.

81

Interface utilisateur (1)

- La conception de l'interface utilisateur est un travail multidisciplinaire:
  - Les ergonomistes/psychologue industriel.
  - Les concepteurs du logiciel.
  - Les programmeurs informaticiens.
  - Les clients.
- Néanmoins, il existe des règles simples qu'il faut toujours respecter:
  - Placer l'utilisateur au centre de la préoccupation.
  - Réduire la mémorisation de la part de l'utilisateur.
  - Créer une interface qui est consistante.

Interface utilisateur (2)

- La conception de l'interface implique généralement 4 modèles:
  - Le modèle de conception
    - C'est l'aspect informatique lié à la création et à la programmation de l'interface utilisateur (souvent graphique).
  - Le modèle utilisateur
    - Ce modèle présente le profile de l'utilisateur du système logiciel.
    - Généralement on doit tenir compte du niveau novice, intermédiaire et avancé des connaissances de l'utilisateur.

Interface utilisateur (3)

- La conception de l'interface implique généralement 4 modèles (suite):
  - Le modèle de perception
    - C'est l'image mentale du système logiciel dans l'esprit de l'utilisateur.
    - L'utilisateur construit cette image à partir de son expérience avec le système.
  - Le modèle image du système
    - Le « look and feel » de l'interface graphique.
    - L'explication dans le manuel d'utilisateur.

L 'utilisateur accepterait plus facilement le logiciel si le modèle de perception et l'image du système coïncident.

Interface utilisateur (4)

- La liste des activités pour la conception de l'interface utilisateur est:
  - Analyse du profil usager
    - Quels types d'utilisateurs utiliseront le logiciel ?
    - Quelles sont ses caractéristiques ?
  - Analyse des tâches
    - Quelles sont les tâches que l'utilisateur peuvent accomplir ?
    - Quelle est la séquence d'actions nécessaire pour accomplir ces tâches (une séquence par tâche) ?
  - Définir les composants d'interface
    - Interface graphique → définir les « contrôles »

Interface utilisateur (5)

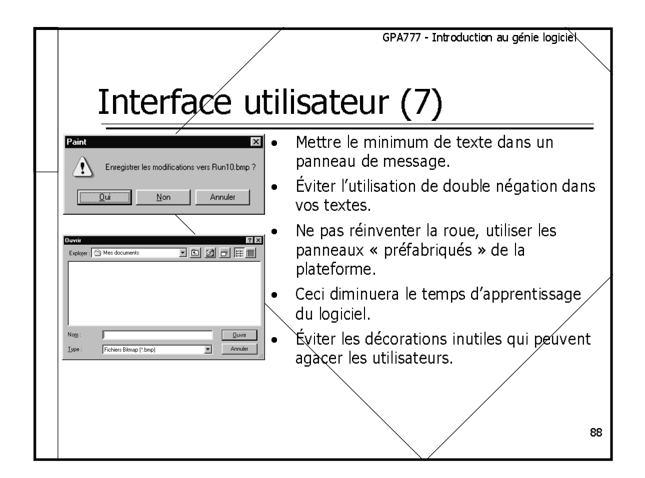
- Définir les composants d'interface (suite)
  - Interface non graphique → définir la syntaxe et la sémantique des commandes.
- Construction de l'interface
  - Interface graphique → utiliser la capacité de l'environnement de développement (Ex: resource editor de VC++).
  - Interface non graphique → création de l'interpréteur de commandes.
- Validation de l'interface
  - Utilisation de groupes cibles (alpha, beta testers).
  - Utilisation des modèles psychologiques.

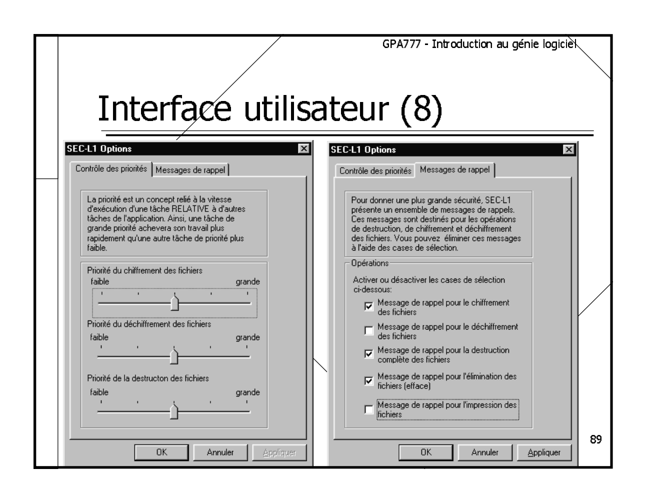
Interface utilisateur (6)

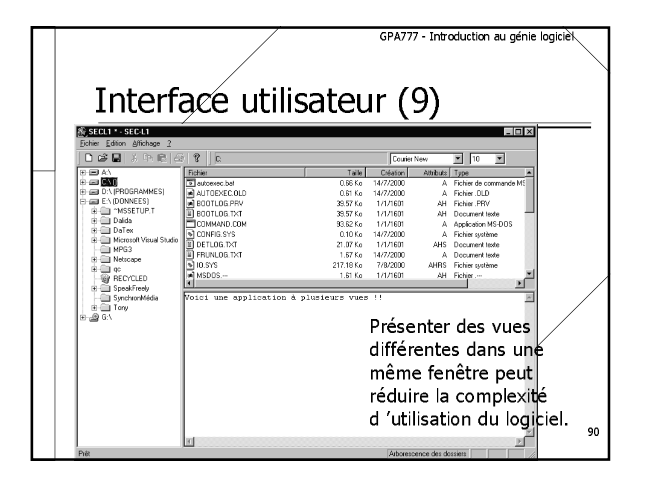
- La plupart des applications modernes utilisent une interface utilisateur qui est graphique (GUI).
- Voici quelques recommandations sur l'utilisation des éléments graphiques.

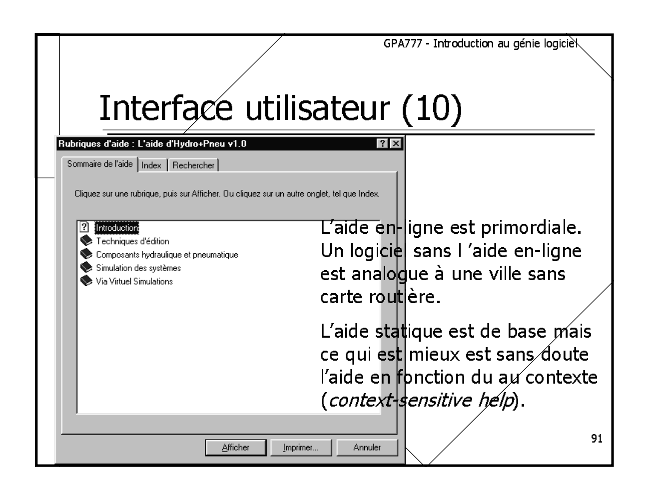


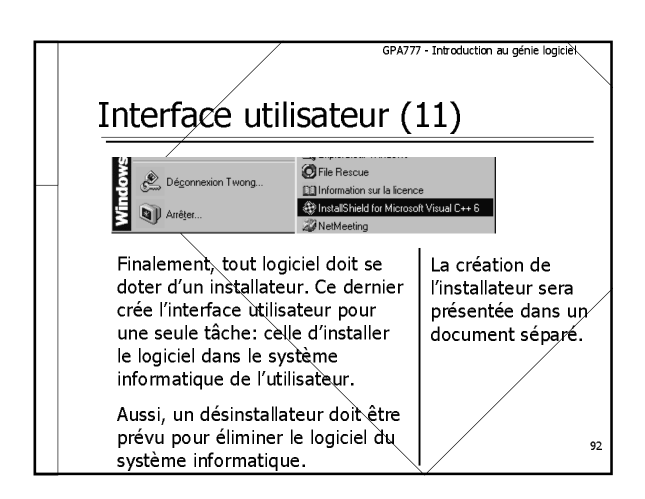
•Le contrôle Bouton doit toujours enclencher une **action visible**. Sans quoi l'utilisateur peut créer une image mentale incorrecte du système.
•Utiliser une liste déroulante avec zone d'édition si cette dernière peut réellement faciliter la tâche. Si non, rendre cette zone non éditable.
•Le contrôle barre d'outils peut être statique ou flottante. Dans ce dernier cas, prévoir une façon de la faire réapparaître après sa fermeture.

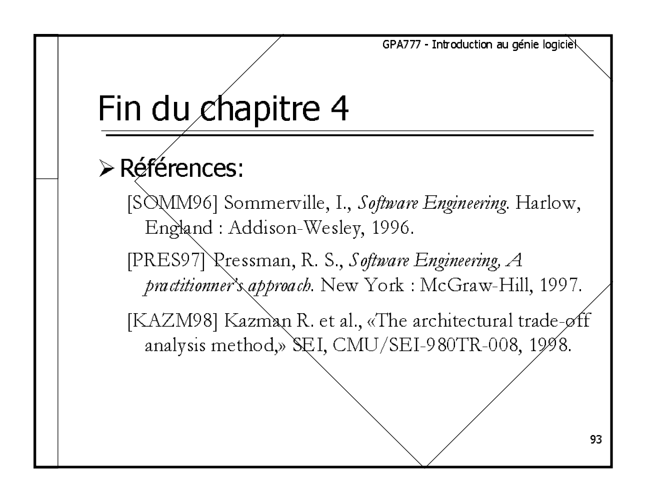

GPA777 - Introduction au génie logiciel
Interface utilisateur (7)
Paint
Enregistrer les modifications vers Run10.bmp ?
Oui
Non
Annuler
Ouvrir
Nom :
Type :
Fichiers Bitmap (*.bmp)
Ouvrir
Annuler
• Mettre le minimum de texte dans un panneau de message.
• Éviter l'utilisation de double négation dans vos textes.
• Ne pas réinventer la roue, utiliser les panneaux « préfabriqués » de la plateforme.
• Ceci diminuera le temps d'apprentissage du logiciel.
• Éviter les décorations inutiles qui peuvent agacer les utilisateurs.
88


GPA777 - Introduction au génie logiciel
Interface utilisateur (8)
SEC-L1 Options
Contrôle des priorités
Messages de rappel
La priorité est un concept relié à la vitesse d'exécution d'une tâche RELATIVE à d'autres tâches de l'application. Ainsi, une tâche de grande priorité achevera son travail plus rapidement qu'une autre tâche de priorité plus faible.
Priorité du chiffrement des fichiers
faible
grande
Priorité du déchiffrement des fichiers
Priorité de la destructon des fichiers
OK
Annuler
Appliquer
Pour donner une plus grande sécurité, SEC-L1 présente un ensemble de messages de rappels. Ces messages sont destinés pour les opérations de destruction, de chiffrement et déchiffrement des fichiers. Vous pouvez éliminer ces messages à l'aide des cases de sélection.
Opérations
Activer ou désactiver les cases de sélection ci-dessous:
Message de rappel pour le chiffrement des fichiers
Message de rappel pour le déchiffrement des fichiers
Message de rappel pour la destruction complète des fichiers
Message de rappel pour l'élimination des fichiers (efface)
Message de rappel pour l'impression des fichiers
89


GPA777 - Introduction au génie logiciel
Interface utilisateur (9)
SECL1 * - SEC-L1
Fichier Edition Affichage ?
Voici une application à plusieurs vues !!
Présenter des vues différentes dans une même fenêtre peut réduire la complexité d 'utilisation du logiciel.
Prêt
Arborescence des dossiers
90


GPA777 - Introduction au génie logiciel
Interface utilisateur (10)
Rubriques d'aide : L'aide d'Hydro+Pneu v1.0
Sommaire de l'aide
Index
Rechercher
Cliquez sur une rubrique, puis sur Afficher. Ou cliquez sur un autre onglet, tel que Index.
Introduction
Techniques d'édition
Composants hydraulique et pneumatique
Simulation des systèmes
Via Virtuel Simulations
Afficher
Imprimer...
Annuler
L'aide en-ligne est primordiale. Un logiciel sans l 'aide en-ligne est analogue à une ville sans carte routière.
L'aide statique est de base mais ce qui est mieux est sans doute l'aide en fonction du au contexte (context-sensitive help).
91


GPA777 - Introduction au génie logiciel
Interface utilisateur (11)
Windows
Déconnexion Twong...
Arrêter...
File Rescue
Information sur la licence
InstallShield for Microsoft Visual C++ 6
NetMeeting
Finalement, tout logiciel doit se doter d'un installateur. Ce dernier crée l'interface utilisateur pour une seule tâche: celle d'installer le logiciel dans le système informatique de l'utilisateur.
Aussi, un désinstallateur doit être prévu pour éliminer le logiciel du système informatique.
La création de l'installateur sera présentée dans un document séparé.
92


GPA777 - Introduction au génie logiciel
Fin du chapitre 4
Références:
[SOMM96] Sommerville, I., Software Engineering. Harlow, England : Addison-Wesley, 1996.
[PRES97] Pressman, R. S., Software Engineering, A practitionner's approach. New York : McGraw-Hill, 1997.
[KAZM98] Kazman R. et al., «The architectural trade-off analysis method,» SEI, CMU/SEI-980TR-008, 1998.
93